Matemáticas

Trabajo Integrador 2: Matemática y Programación

Integrantes:

- **DNI 1:** 34565164
- **DNI 2:** 43122514
- **DNI 3:** 42892905
- **DNI 4:** 41785314

Link Video Grupal: https://youtu.be/AQJZv70D88o

Enfoque: Conjuntos y Operaciones Básicas

Parte 1 – Desarrollo Matemático (Conjuntos y Lógica)

1. Conjuntos de Dígitos Únicos por DNI

A partir de los DNIs, formamos los siguientes conjuntos de dígitos únicos:

- **Conjunto A (DNI 34565164):** A={1,3,4,5,6}
- **Conjunto B (DNI 43122514):** B={1,2,3,4,5}
- **Conjunto C (DNI 42892905):** C={0,2,4,5,8,9}
- **Conjunto D (DNI 41785314):** D={1,3,4,5,7,8}

2. Operaciones entre Conjuntos

Unión (∪)

- **A ∪ B:** {1,2,3,4,5,6}
- **A ∪ C:** {0,1,2,3,4,5,6,8,9}
- **A ∪ D:** {1,3,4,5,6,7,8}
- **B ∪ C:** {0,1,2,3,4,5,8,9}
- **B ∪ D:** {1,2,3,4,5,7,8}
- **C ∪ D:** {0,1,2,3,4,5,7,8,9}

Intersección (∩)

- **A ∩ B:** {1,3,4,5}
- **A ∩ C:** {4,5}
- **A ∩ D:** {1,3,4,5}
- **B ∩ C:** {2,4,5}
- **B ∩ D:** {1,3,4,5}
- **C ∩ D:** {4,5,8}

Diferencia (−)

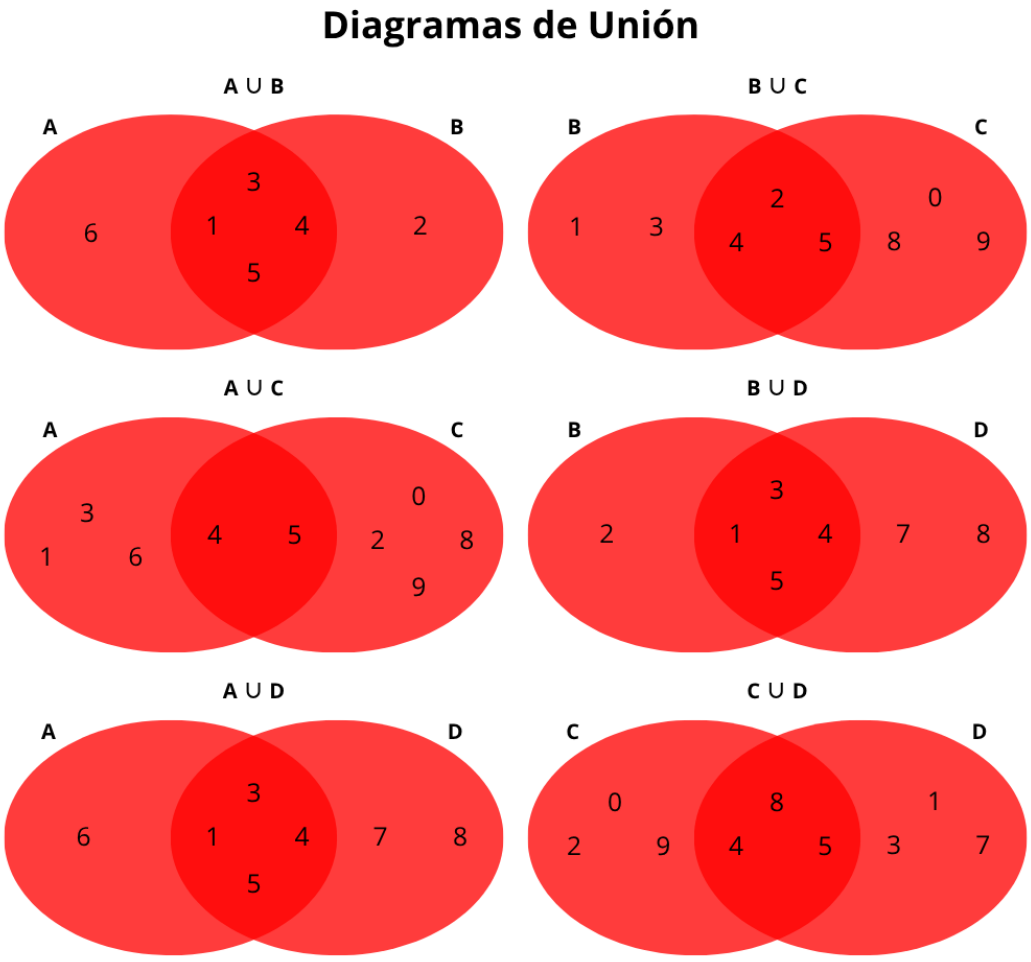
- **A - B:** {6}
- **B - A:** {2}
- **A - C:** {1,3,6}
- **C - A:** {0,2,8,9}
- **A - D:** {6}
- **D - A:** {7,8}
- **B - C:** {1,3}
- **C - B:** {0,8,9}
- **B - D:** {2}
- **D - B:** {7,8}
- **C - D:** {0,2,9}
- **D - C:** {1,3,7}

Diferencia Simétrica (Δ)

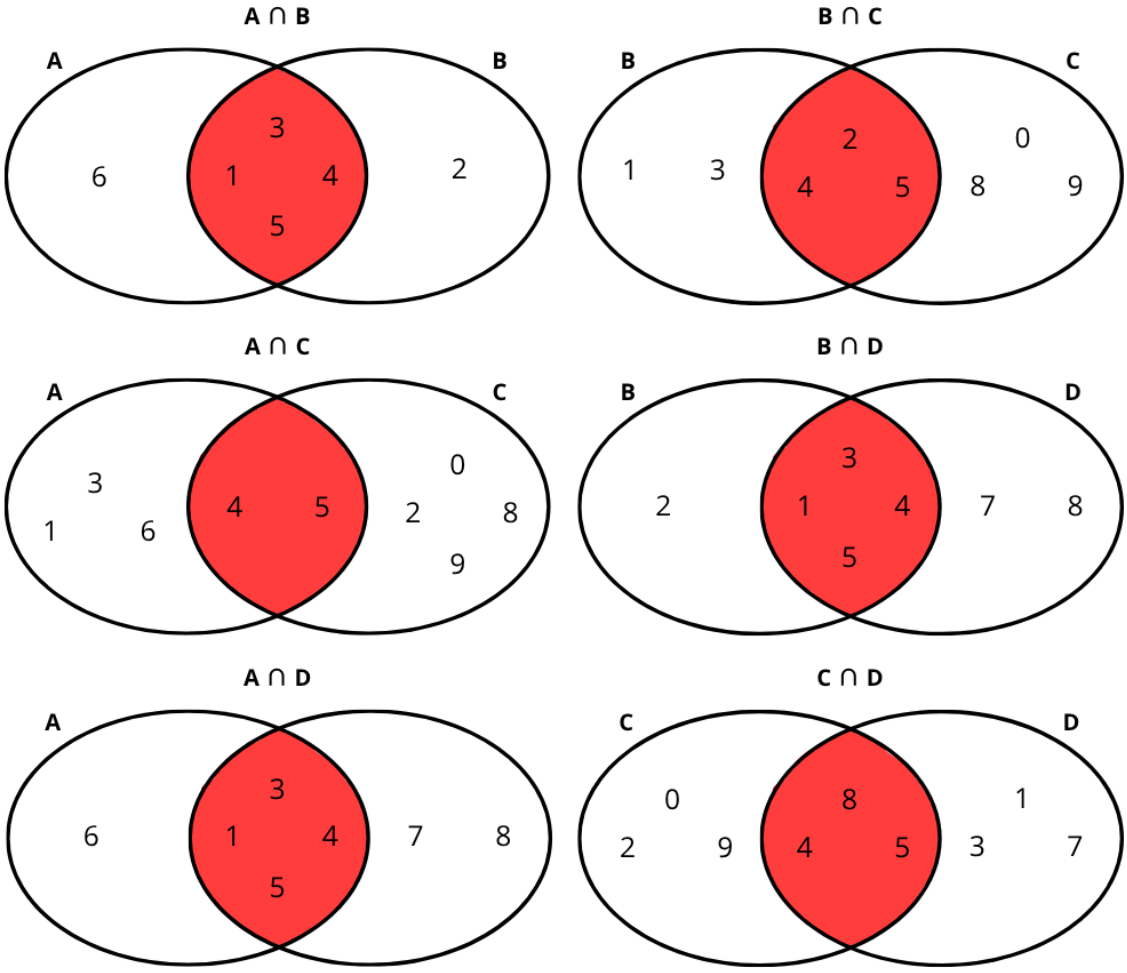
- **A Δ B:** (A−B)∪(B−A)={6}∪{2}={2,6}
- **A Δ C:** (A−C)∪(C−A)={1,3,6}∪{0,2,8,9}={0,1,2,3,6,8,9}
- **A Δ D:** (A−D)∪(D−A)={6}∪{7,8}={6,7,8}
- **B Δ C:** (B−C)∪(C−B)={1,3}∪{0,8,9}={0,1,3,8,9}

- **B Δ D:** (B−D)U(D−B)={2}U{7,8}={2,7,8}
- **C Δ D:** (C−D)U(D−C)={0,2,9}U{1,3,7}={0,1,2,3,7,9}

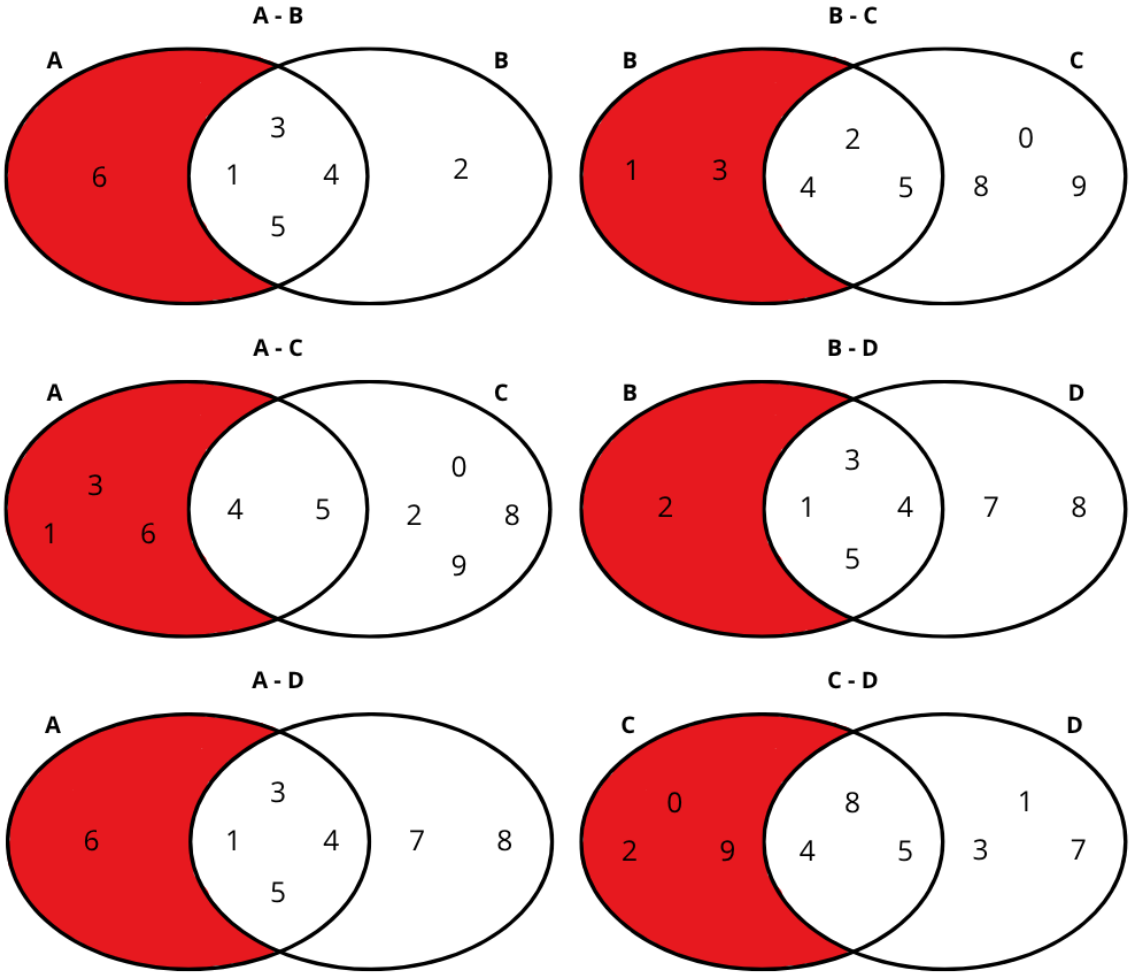
3. Diagramas de Venn



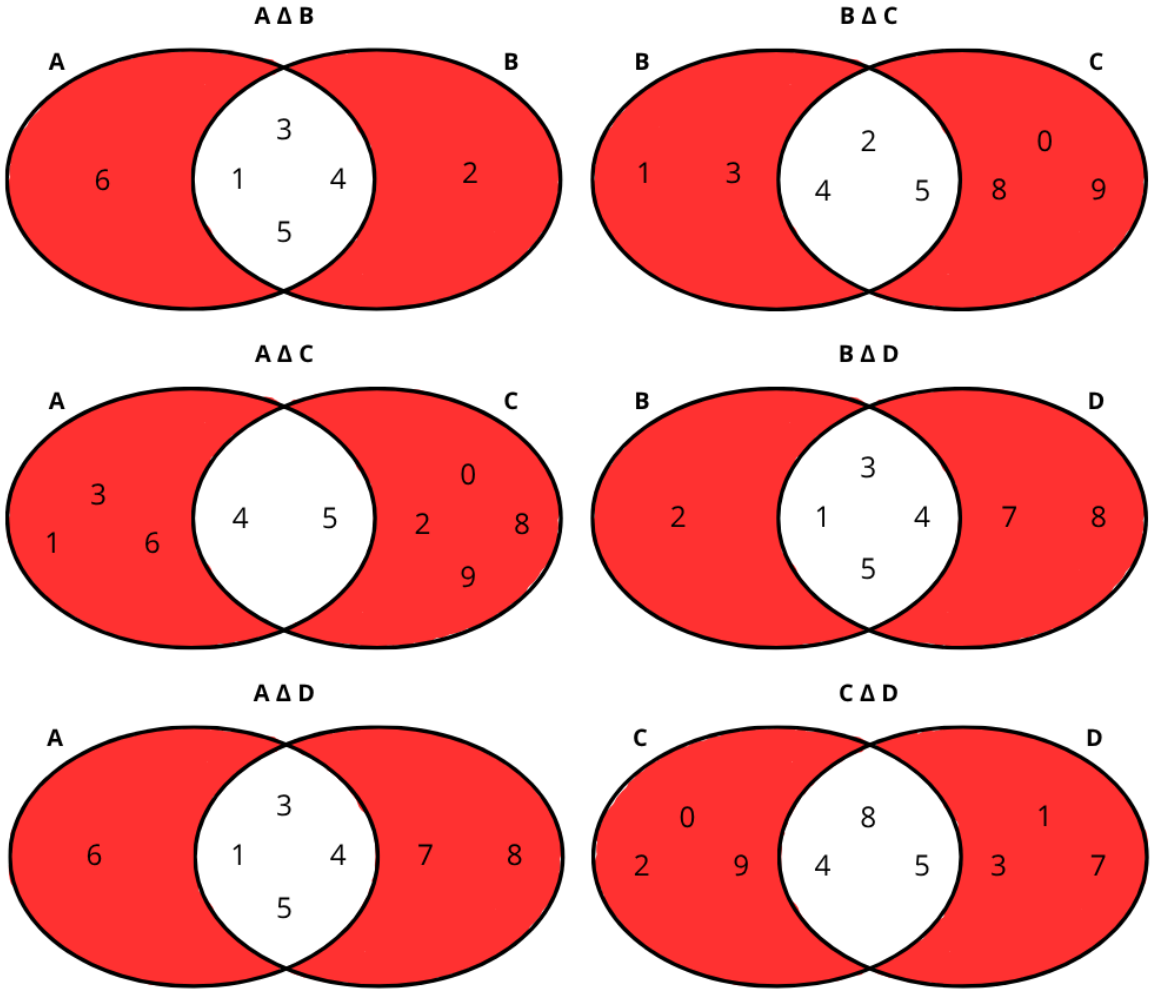
Diagramas de Intersección



Diagramas de Diferencia



Diagramas de Diferencia Simétrica



4. Expresiones Lógicas en Lenguaje Natural

Aquí se presentan dos expresiones lógicas que pueden implementarse en Python, junto con el resultado esperado para los conjuntos del grupo:

1. **Expresión Lógica 1:** "Si la unión de todos los conjuntos tiene menos de 10 elementos únicos Y ningún conjunto individual contiene el dígito 0, entonces se considera un 'Grupo con Números Limitados y Sin Ceros'."

- **Resultado con los conjuntos del grupo:**
  - Unión de todos los conjuntos (AUBUCUD): {0,1,2,3,4,5,6,7,8,9} (10 elementos únicos).
  - El Conjunto C contiene el dígito 0.
  - Dado que la unión tiene 10 elementos y el Conjunto C contiene el 0, la condición completa es **Falsa**.

2. **Expresión Lógica 2:** "Si la intersección de al menos dos conjuntos distintos tiene exactamente 4 elementos, Y el conjunto con la mayor cantidad de elementos contiene al menos un número par y un número impar, entonces se cumple la 'Condición de Diversidad Interconectada'."

   - **Resultado con los conjuntos del grupo:**
     - **Intersecciones con 4 elementos:**
       - A∩B={1,3,4,5} (4 elementos)
       - A∩D={1,3,4,5} (4 elementos)
       - B∩D={1,3,4,5} (4 elementos)
       - Se cumple la primera parte de la condición.
     - **Conjunto con la mayor cantidad de elementos:**
       - Conjunto A: 5 elementos
       - Conjunto B: 5 elementos
       - Conjunto C: 6 elementos
       - Conjunto D: 6 elementos
       - Los conjuntos C y D tienen la mayor cantidad de elementos (6).
       - **Conjunto C ({0,2,4,5,8,9}):** Contiene números pares (0, 2, 4, 8) y números impares (5, 9).
       - **Conjunto D ({1,3,4,5,7,8}):** Contiene números pares (4, 8) y números impares (1, 3, 5, 7).
       - Se cumple la segunda parte de la condición.
     - Por lo tanto, la condición completa es **Verdadera**.

Parte 2 – Desarrollo del Programa en Python

Explicación del Código

1. **obtener_digitos_unicos(numero_dni):**
    - Esta función toma un número de DNI como entrada.
    - Convierte el número DNI a una cadena de texto para poder iterar sobre sus dígitos.
    - Utiliza una **comprensión de conjunto** (set(int(digito) for digito in str(numero_dni))) para extraer cada dígito, convertirlo a entero y agregarlo automáticamente a un conjunto, eliminando cualquier duplicado.
    - Retorna el conjunto de dígitos únicos.
2. **contar_frecuencia_digitos(numero_dni):**
    - Esta función también recibe un número de DNI.
    - Inicializa un diccionario vacío llamado frecuencia_digitos.
    - **Itera** sobre cada dígito del DNI (convertido a cadena).
    - Por cada dígito, lo convierte a entero y usa el método get() del diccionario para incrementar su contador. Si el dígito no está en el diccionario, get(digito_int, 0) devuelve 0, y luego se le suma 1.
    - Retorna el diccionario con la frecuencia de cada dígito.
3. **main():**
    - Esta función simula la ejecución de la parte del código asignada al Integrante 1.
    - **Ingreso de DNIs:** Pide al usuario que ingrese los cuatro DNIs del grupo. Incluye un bucle while y un bloque try-except para validar que se ingresen números y que tengan 8 dígitos.
    - **Generación de Conjuntos:** Llama a obtener_digitos_unicos para cada DNI ingresado y almacena los conjuntos resultantes en una lista.
    - **Operaciones de Unión e Intersección:**
        - Utiliza dos bucles for anidados para iterar sobre todos los pares únicos de conjuntos (evitando comparaciones repetidas como A con B y luego B con A, o un conjunto consigo mismo).

- Aplica los métodos .union() y .intersection() de los objetos set de Python, que realizan estas operaciones de manera eficiente.
- Imprime los resultados de cada operación de forma clara.

- **Conteo de Frecuencia:** Itera sobre la lista de DNIs y llama a contar_frecuencia_digitos para cada uno, mostrando las frecuencias obtenidas.

Enfoque: Frecuencia y Condiciones Simples

Parte 1 – Desarrollo Matemático

1. **Revisión de conjuntos de dígitos únicos**

   Primero, revisamos los conjuntos que se formaron a partir de cada DNI, para asegurarnos de que sólo tuvieran los dígitos sin repetirlos. Por ejemplo:

   - DNI 34565164 nos dio el conjunto A = {1, 3, 4, 5, 6}
   - DNI 43122514 nos dio el conjunto B = {1, 2, 3, 4, 5}
   - DNI 42892905 nos dio el conjunto C = {0, 2, 4, 5, 8, 9}
   - DNI 41785314 nos dio el conjunto D = {1, 3, 4, 5, 7, 8}

2. **Conteo de la frecuencia de los dígitos por DNI**

   Después, contamos cuántas veces aparece cada dígito en cada DNI. Esto nos ayudó a identificar si había patrones o repeticiones dentro de los números.

3. **Análisis de una condición lógica sencilla**

   Trabajamos en entender y analizar esta condición:

   “Si, al juntar todos los conjuntos, hay menos de 10 dígitos diferentes y además ninguno de los conjuntos contiene el dígito 0, entonces se considera que el grupo tiene números limitados y sin ceros.”

Al juntar todos los conjuntos, vimos que hay exactamente 10 dígitos distintos (0 al 9), por lo que no se cumple la primera parte. Además, uno de los conjuntos sí tiene un cero (el conjunto C), así que tampoco se cumple la segunda parte.

Por lo tanto, la condición es falsa para estos DNIs.

Parte 2 – Desarrollo del Programa en Python

Una parte importante del trabajo fue desarrollar funciones en Python que analicen los DNIs de los integrantes del grupo. En particular, creamos una función llamada **validar_dnis_y_conjuntos,** que se encarga de hacer dos cosas muy concretas:

1. Revisar si alguno de los DNIs tiene el dígito 0.
2. Verificar si entre todos los dígitos de los DNIs hay menos de 10 distintos.

Esto nos sirve para saber si el grupo cumple con una condición que llamamos **"Números Limitados y Sin Ceros"**, es decir, que ninguno tenga el 0 y que no estén todos los dígitos del 0 al 9.

¿Cómo funciona la función?

- **Entrada:** recibe dos cosas:
  - Una lista con los DNIs de los integrantes.
  - Una lista con los conjuntos de dígitos únicos de cada uno de esos DNIs.

- **Paso 1:** Recorre los DNIs y detecta cuáles tienen el dígito 0.
  Por ejemplo, si uno de los DNIs es **42892905**, lo convierte a texto y ve que contiene un 0, así que lo guarda.
- **Paso 2:** Toma todos los conjuntos de dígitos y los une en uno solo.
  Así obtiene todos los dígitos distintos que aparecen entre todos los DNIs.

- **Paso 3:** Comprueba si en total hay menos de 10 dígitos distintos.
  Si hay 10, quiere decir que están todos los dígitos del 0 al 9; si hay menos, entonces faltan algunos.

- **Paso 4:** Muestra por pantalla un resumen con los resultados:

  - Qué DNIs tienen el 0.
  - Qué dígitos están presentes entre todos.

- Si faltan o no dígitos.

- **Salida:** devuelve dos cosas:

  - Una lista con los DNIs que tienen el dígito 0.
  - Un valor booleano (True o False) que indica si la unión de dígitos está incompleta.

Enfoque: Lógica Compleja y Operaciones

Parte 1 – Desarrollo Matemático

Expresiones lógicas en lenguaje natural:

Expresión Lógica 1: "Si la unión de todos los conjuntos tiene menos de 10 elementos únicos Y ningún conjunto individual contiene el dígito 0, entonces se considera un 'Grupo con Números Limitados y Sin Ceros'."

- **Resultado con los conjuntos del grupo:**
  - Unión de todos los conjuntos (AUBUCUD): {0,1,2,3,4,5,6,7,8,9} (10 elementos únicos).
  - El Conjunto C contiene el dígito 0.
  - Dado que la unión tiene 10 elementos y el Conjunto C contiene el 0, la condición completa es **Falsa**.

Expresión Lógica 2: "Si la intersección de al menos dos conjuntos distintos tiene exactamente 4 elementos, Y el conjunto con la mayor cantidad de elementos contiene al menos un número par y un número impar, entonces se cumple la 'Condición de Diversidad Interconectada'."

- **Resultado con los conjuntos del grupo:**
  - **Intersecciones con 4 elementos:**
    - $A \cap B = \{1,3,4,5\}$ (4 elementos)
    - $A \cap D = \{1,3,4,5\}$ (4 elementos)
    - $B \cap D = \{1,3,4,5\}$ (4 elementos)
    - Se cumple la primera parte de la condición.

- **Conjunto con la mayor cantidad de elementos:**
  - Conjunto A: 5 elementos
  - Conjunto B: 5 elementos
  - Conjunto C: 6 elementos
  - Conjunto D: 6 elementos
  - Los conjuntos C y D tienen la mayor cantidad de elementos (6).
  - **Conjunto C ({0,2,4,5,8,9}):** Contiene números pares (0, 2, 4, 8) y números impares (5, 9).
  - **Conjunto D ({1,3,4,5,7,8}):** Contiene números pares (4, 8) y números impares (1, 3, 5, 7).
  - Se cumple la segunda parte de la condición. Por lo tanto, la condición completa es **Verdadera**.

Verificación de los diagramas de Venn:

1. Diagramas de unión: debe cubrir todos los elementos presentes en cualquiera de los conjuntos.

- **A ∪ B:** el diagrama muestra: {1, 2, 3, 4, 5, 6} (todo sombreado).
- **B ∪ C:** el diagrama muestra: {0, 1, 2, 3, 4, 5, 8, 9} (todo sombreado).
- **A ∪ C:** el diagrama muestra: {0, 1, 2, 3, 4, 5, 6, 8, 9} (todo sombreado).
- **B ∪ D:** el diagrama muestra: {1, 2, 3, 4, 5, 7, 8} (todo sombreado).
- **A ∪ D:** el diagrama muestra: {1, 3, 4, 5, 6, 7, 8} (todo sombreado).
- **C ∪ D:** el diagrama muestra: {0, 1, 2, 3, 4, 5, 7, 8, 9} (todo sombreado).

2. Diagramas de intersección: el área sombreada debe cubrir sólo los elementos comunes a ambos conjuntos.

   - **A ∩ B: e**l diagrama muestra: {1, 3, 4, 5} (solo la intersección sombreada).
   - **B ∩ C:** el diagrama muestra: {2, 4, 5} (solo la intersección sombreada).
   - **A ∩ C: e**l diagrama muestra: {4, 5} (solo la intersección sombreada).
   - **B ∩ D: e**l diagrama muestra: {1, 3, 4, 5} (solo la intersección sombreada).
   - **A ∩ D: e**l diagrama muestra: {1, 3, 4, 5} (solo la intersección sombreada).
   - **C ∩ D: e**l diagrama muestra: {4, 5, 8} (solo la intersección sombreada).

3. Diagramas de diferencia: el área sombreada debe cubrir los elementos del primer conjunto que no están en el segundo.
   - **A - B: e**l diagrama muestra: {6} (solo la parte de A que no intersecta con B sombreada).
   - **B - C: e**l diagrama muestra: {1, 3} (solo la parte de B que no intersecta con C sombreada).
   - **A - C: e**l diagrama muestra: {1, 3, 6} (solo la parte de A que no intersecta con C sombreada).
   - **B - D: e**l diagrama muestra: {2} (solo la parte de B que no intersecta con D sombreada).
   - **A - D: e**l diagrama muestra: {6} (solo la parte de A que no intersecta con D sombreada).
   - **C - D: e**l diagrama muestra: {0, 2, 9} (solo la parte de C que no intersecta con D sombreada).

4. Diagramas de diferencia simétrica: el área sombreada debe cubrir los elementos que están en cualquiera de los dos conjuntos, pero no en su intersección.
   - **A Δ B:** el diagrama muestra: {2, 6} (sombreadas las partes no intersecantes de A y B).

- **B Δ C:** el diagrama muestra: {0, 1, 3, 8, 9} (sombreadas las partes no intersecantes de B y C).
- **A Δ C:** el diagrama muestra: {0, 1, 2, 3, 6, 8, 9} (sombreadas las partes no intersecantes de A y C).
- **B Δ D:** el diagrama muestra: {2, 7, 8} (sombreadas las partes no intersecantes de B y D).
- **A Δ D:** el diagrama muestra: {6, 7, 8} (sombreadas las partes no intersecantes de A y D).
- **C Δ D:** el diagrama muestra: {0, 1, 2, 3, 7, 9} (sombreadas las partes no intersecantes de C y D).

Parte 2 – Desarrollo del programa en Python

En esta sección, nuestro enfoque principal fue la implementación de la lógica más compleja del proyecto, especialmente la evaluación de expresiones lógicas avanzadas con los conjuntos de DNI y el análisis de los años de nacimiento del grupo.

Evaluación de Expresiones Lógicas Complejas (con conjuntos de DNI)

Desarrollamos el código para evaluar dos condiciones lógicas específicas que reflejan la diversidad y la interconexión de los dígitos en los DNI del grupo.

1. Evaluación de la "Expresión Lógica 2: Condición de diversidad interconectada" Esta expresión es un Y lógico (AND) de dos sub-condiciones. El programa determina si se cumplen ambas para considerar la expresión como verdadera.

Funcionamiento:

- **Entrada:** se utilizan los conjuntos de dígitos únicos previamente generados a partir de los DNIs de todos los integrantes (Conjunto A, B, C y D).

- **Paso 1: verificar la cardinalidad de intersecciones:**
    - El programa itera a través de todos los posibles pares de conjuntos de DNI (A con B, A con C, B con C, etc.).
    - Para cada par, calcula su intersección.
    - Cuenta cuántas de estas intersecciones resultan tener 4 elementos.
    - La primera parte de la expresión se considera verdadera si se encuentran al menos dos de estas intersecciones con 4 elementos.
- **Paso 2: analizar los conjuntos con mayor cardinalidad:**
    - El programa primero identifica cuál es el tamaño máximo que tiene cualquiera de los conjuntos de DNI.
    - Luego, identifica a todos los conjuntos que alcanzan esa cardinalidad máxima.
    - Para cada uno de estos conjuntos "más grandes", el programa verifica si contiene al menos un dígito par y al menos un dígito impar.
    - La segunda parte de la expresión se considera verdadera si al menos uno de esos conjuntos con mayor cardinalidad cumple con tener dígitos pares e impares.
- **Paso 3: conclusión de la expresión lógica 2:**
    - Finalmente, la "Condición de Diversidad Interconectada" se cumple si tanto la primera parte (de las intersecciones con 4 elementos) como la segunda parte (de los conjuntos grandes con pares e impares) resultan ser verdaderas.
    - El programa imprime el resultado final (True o False) y lo compara con el resultado esperado que determinamos en la fase matemática (en nuestro caso, esperábamos True).

2. Evaluación de una "Expresión lógica adicional: diversidad numérica alta" Esta es otra condición lógica que el programa evalúa, también como un y (AND) de dos sub-condiciones.

Funcionamiento:

- **Entrada:** Se utilizan específicamente el conjunto de dígitos del DNI del Integrante 3 (conjunto C) y del Integrante 4 (conjunto D).

- **Paso 1: cardinalidad del integrante 3:**
    - El programa verifica directamente si la cantidad de dígitos únicos en el conjunto C es mayor a 5.
- **Paso 2: presencia del dígito 0 en el integrante 4:**
    - El programa comprueba si el conjunto D no contiene el dígito 0.
- **Paso 3: conclusión de la expresión adicional:**
    - La "Diversidad Numérica Alta" se cumple si la cardinalidad de mi conjunto es mayor a 5 y el conjunto del Integrante 4 no tiene el dígito 0.
    - El programa imprime el resultado final (True o False) y lo compara con el resultado esperado que calculamos.

Análisis de Años de Nacimiento del Grupo

Además de la lógica de DNI, se realizó la implementación de funcionalidades para analizar los años de nacimiento de los integrantes.

Funcionamiento:

- **Entrada:** El programa solicita al usuario que ingrese, uno por uno, el año de nacimiento de cada integrante del grupo. Se incluye una validación básica para asegurar que el año ingresado sea un número válido.

- **Paso 1: conteo de años pares e impares:**
    - Una vez que se tienen todos los años de nacimiento, el programa itera a través de ellos.
    - Para cada año, utiliza una operación matemática simple (el módulo divisor % 2) para determinar si es un año par o impar.
    - Mantiene dos contadores separados: uno para los años pares y otro para los impares, incrementándose según corresponda.
    - Finalmente, imprime el total de nacimientos en años pares y el total en años impares.

- **Paso 2: determinación del "Grupo Z":**
    - El programa parte de la suposición inicial de que el grupo sí es un "Grupo Z" (es decir, que todos nacieron después del año 2000).
    - Luego, revisa cada año de nacimiento. Si en algún momento encuentra un solo integrante que haya nacido en el año 2000 o antes, automáticamente el grupo deja de ser un "Grupo Z" y el programa detiene la verificación.
    - Si revisa todos los años y no encuentra a nadie que haya nacido en el 2000 o antes, entonces confirma que la condición de "Grupo Z" se cumple.
    - El programa imprime el resultado final, indicando si el grupo cumple o no la condición de "Grupo Z".

Enfoque: Funciones Específicas y Producto Cartesiano

Parte 1 – Desarrollo Matemático

En esta sección nos vamos a enfocar en el desarrollo de una función en particular, que se va a encargar de que determinar si un integrante del grupo nació en un año bisiesto. Para esto vamos a comenzar con analizar matemáticamente que es un año bisiesto y que lo caracteriza.

- Un año bisiesto es divisible por 4 y también debe ser divisible por 100 y por 400, si no se cumplen estas condiciones no es un año bisiesto.

    - 2000 es bisiesto (divisible por 4 y por 400)
    - 1900 no es bisiesto (divisible por 100 pero no por 400)
    - 2024 es bisiesto (divisible por 4 y no por 100)

Otra implementación al programa fue el de producto cartesiano, para esto utilizaremos como dos conjuntos al año(A) y a la edad(B), representándose como:

Año X Edad

Esto quiere decir **Todas las combinaciones posibles** donde el primer elemento viene de A y el segundo de B.

En nuestro caso supongamos que tenemos estos dos conjuntos

A = {26,25}

B = {1999, 2000}

Entonces el producto cartesiano *A x B* es:

A x B = {(26,1999),(26,2000),(25,1999),(25,2000)}

Cada elemento de A, se combina con todos los elementos de B.

Parte 2 – Desarrollo del programa en Python

- **es_bisiesto:**
  - En cuanto al desarrollo de la función, a partir del año de nacimiento de cada integrante determinaremos si se cumplen las condiciones mencionadas. Se recorre todos los años de nacimiento del grupo y se verifica si alguno de ellos es bisiesto, si lo es la función devuelve **True.**

- **calcular_edad:**
  - Para función de cálculo de la edad actual, se **importa la clase datetime del módulo datetime**. La cual utilizaremos para acceder a la fecha y hora actual del sistema. Y nuestro caso lo implementamos para poder acceder al año actual, utilizando la función **date.today().year.**

    - Por ejemplo, si alguien nació en 1999 y estamos en 2025:
      Edad = 2025 − 1999 = 26 años

    Esta función se aplica a todos los años de nacimiento ingresados, y devuelve una lista con las edades actuales de cada persona del grupo.

- **Producto Cartesiano:**
  - Para realizar el producto cartesiano entre nuestra lista de Año y la de Edad, utilizaremos la función importada de **product** del modelo **itertools**, esta nos va a ayudar a calcular el producto cartesiano entre dos listas o conjuntos. Esto nos va a generar **todas las combinaciones posibles** entre los años y las edades.

  - Ejemplo:

```
from itertools import product

años = [2000, 2001]

edades = [23, 24]

resultado = list(product(años, edades))

print(resultado)


[(2000, 23), (2000, 24), (2001, 23), (2001, 24)]
```